Rami Pellumbi

Final Project Report

Introduction

The purpose of this project was to perform automatic segmentations of a dataset of middle coronal slices extracted from an MRI image. While most of the work for the final project was implemented in milestone 2, a considerable amount of changes were made to achieve better automatic segmentation results. For change implementations I recommend the included code is read as it is well commented.

Changes Made

The changes made were made with one goal in mind: to increase the average Jaccard index between the most frequent training label computed for a set of training images mapped onto a validation image against the validation image itself. Doing this would ensure that we would have a better automatic segmentation of the testing images which we had nothing to compare to.

Transformer

The first change made to the geometric registration function was to add more parameters. Now instead of a global scalar there is a scalar along both the x and y coordinate. A shear variable was also added. Thus the registration is now a 6 parameter transformation consisting of parameters (scalex, scaley, rotate, translatex, translatey, shear).

To do the transformation in milestone 2 cv2's affine transform function was used. This yielded good results but was a bit tedious since the matrices had to be made manually. The transformer in the final project was thus switched to skimages warp function. It made it easier to implement all the parameters..

In regards to interpolation, bilinear was used instead of nearest neighbor. This yielded the best results out of all interpolation methods. The method for zero padding was also switched to 'symmetric' instead of 'constant' for better results.

Loss Function

The loss function was switched from measuring mean squared error (MSE) to structural similarity measure (SSIM). The reason for doing this was that MSE has an issue where large difference between pixel values doesn't necessarily mean the images are drastically different. It only accounts for perceived errors. SSIM on the other hand attempts to model perceived changes in the image information by comparing smaller subsamples instead of the whole image itself. A built in SSIM function from skimage was used. While it was a lot slower to run it yielded WAY better results.

Note: Since the most similar image is a value of 1 in SSIM the loss function returns -SSIM since our optimizer is trying to find a minimum.

Optimizer

The optimizer changed from using fmin to 'BFGS' in scipy's optimize/minimize module. This was found via trial and error between all available methods that didn't need a Jacobian.

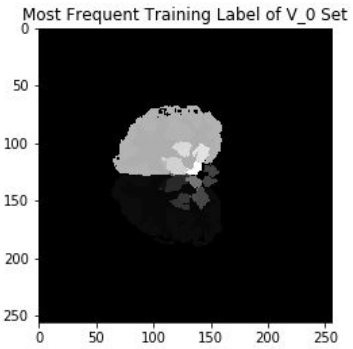
Basinhopping was also attempted but it didn't yield significantly better results and had an absolutely abysmal runtime.

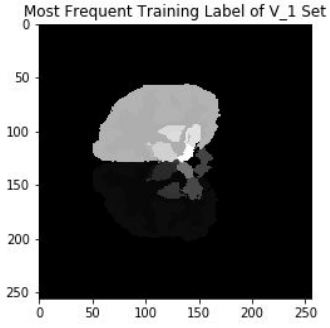
Most Frequent Training Label

The frequent training label employed before worked as follows:

1. Each training image was mapped onto a testing image
    a. Thus we had 6 training images mapped onto one testing image
2. For this set of training images (*j* total) mapped onto one testing image, stack the images on top of each other so that there are *j* images in a stack -i.e each pixel *(m,n)* lined up with all other images
3. Take the mode of each pixel *(m,n)* between the *j* images
    a. If there is a tie return the mode with the lowest value and put this at location *(m,n)* on the output grid

This returned results like so (training images mapped onto validation grid shown here)



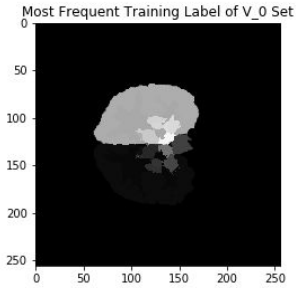


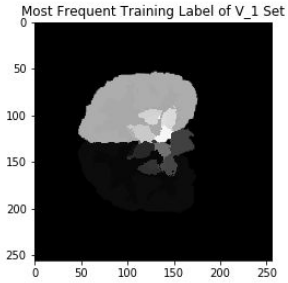
As is visible in the images, the edges are very spotty due to the optimizer not being perfect. To fix this some slight modifications were made to the original algorithm.

The modified algorithm worked as follows:

1. Each training image was mapped onto a testing image
    a. Thus we had 6 training images mapped onto one testing image
2. For this set of training images (*j* total) mapped onto one testing image, stack the images on top of each other so that there are *j* images in a stack -i.e each pixel *(m,n)* lined up with all other images
3. Take the mode of each pixel *(m,n)* between the *j* images
    a. If there is a tie return the mode with the highest value
        i. If the mode at pixel (*m,n)* between the *j* images returned is zero, but at least one image *j* had a nonzero value at pixel *(m.n)* return the mode between the non zero values. Map this to pixel location *(m,n)* on the output grid
        ii. Else map the mode onto pixel *(m,n)* on the output grid

The modified algorithm had the following results:





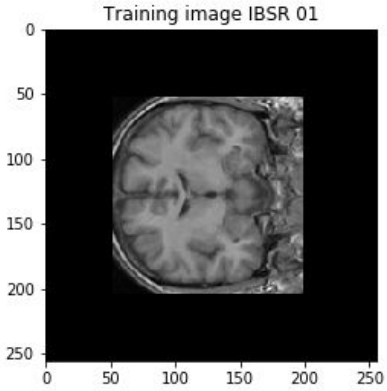
The reason for implementing the following was under the assumption the optimizer was not perfect. So the images did not perfectly overlap. And so the boundaries had an issue where there could be multiple with nothing at that pixel value and multiple with something there. While it is bold to assume this helps segmentation results, it did indeed help performance.

Adding Validation Images to Training Set

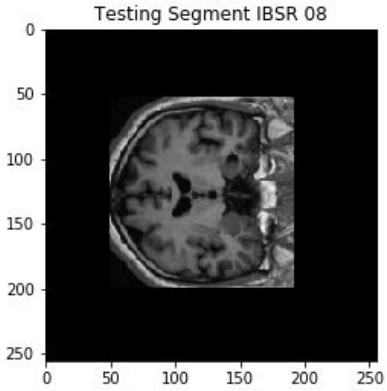
Since we were now mapping training images onto testing images and not validation images the validation images were added to the training data set. This allowed for a larger training set and therefore better results (more things to average over).

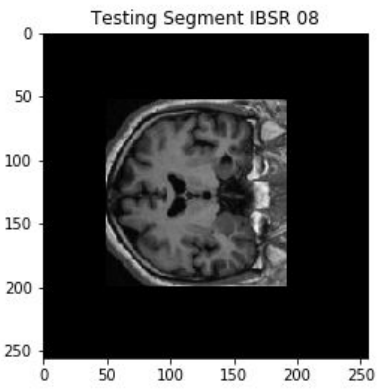
Results

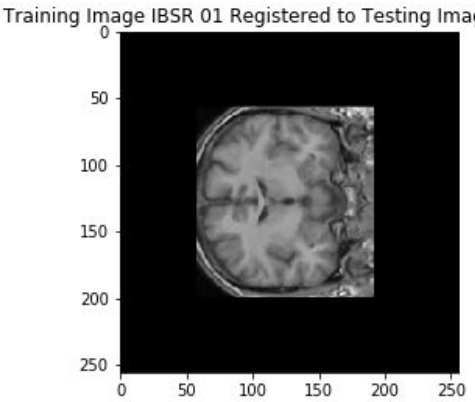
The following shows a training image, the testing image it was mapped onto, the transformed training image, and those same optimal parameters used on the training segment. (View images from top to bottom)

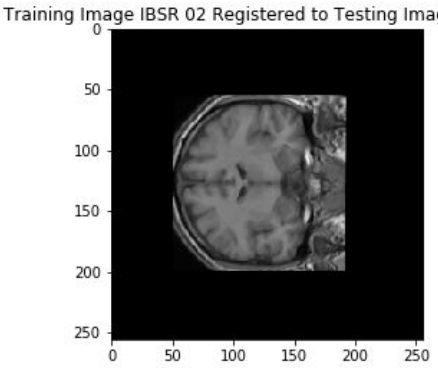

Training image IBSR 01

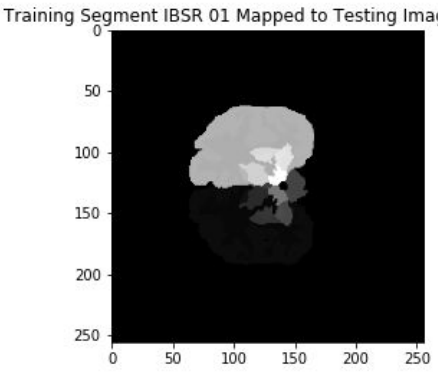

Training image IBSR 02

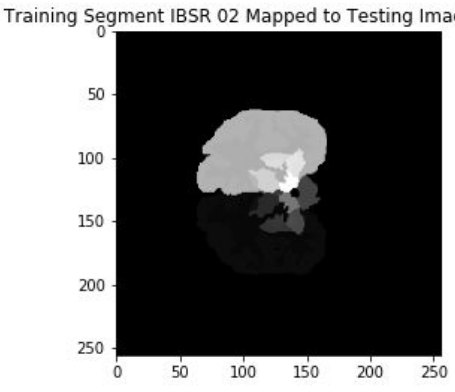

Testing Segment IBSR 08


Testing Segment IBSR 08

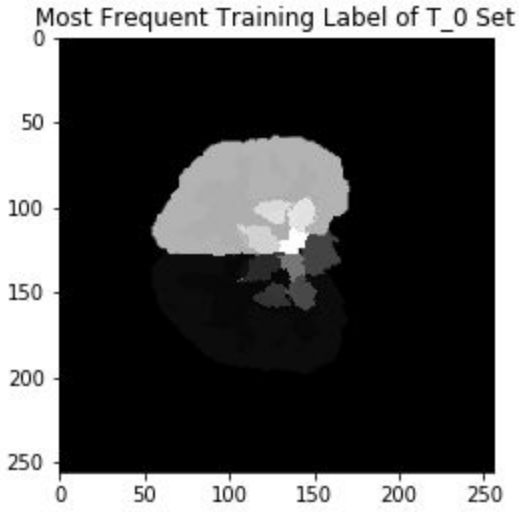

Training Image IBSR 01 Registered to Testing Image IBSR 08

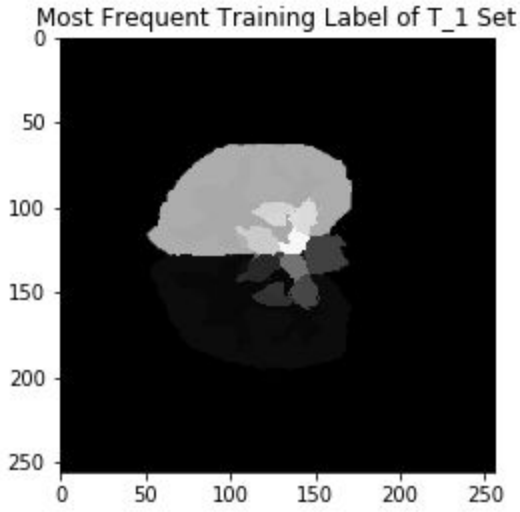

Training Image IBSR 02 Registered to Testing Image IBSR 08


Training Segment IBSR 01 Mapped to Testing Image IBSR 08


Training Segment IBSR 02 Mapped to Testing Image IBSR 08

Below are a few of the most frequent training labels of the training images mapped on testing images IBSR 08 and IBSR 09 respectively.





Conclusion

In conclusion, reasonable results were achieved - with a final dice score of 0.73. One thing that could have been added for increased accuracy in auto segmentation are refining the most frequent training label to do patch based similarity between images. A better fusion label strategy would be the best way to increase automatic segmentation results. If I had to make a hypothesis on what would help the most with the data given this would be it. I just didn't really have a good way to go about this so I stuck with a (what I assume to be inefficient) method of taking the "modified" mode between images. Overall, I am content with the results.